The Report Committee for Dylan Francis Bray
certifies that this is the approved version of the following report:

The MoveSMART Physical Activity Web Game: An Experience Report

SUPERVISING COMMITTEE:

Christine Julien, Supervisor

Darla Castelli

The MoveSMART Physical Activity Web Game: An Experience Report

by

Dylan Francis Bray

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2020

Acknowledgments

I must first thank Dr. Julien for allowing me and my senior design team to take on MoveSMART in the first place, when we were frantically looking for a project to work on. I must also thank those same senior design team members for all of their hard work, without which there would be no work to build from; thank you Grayson Barrett, Kevin Brill, Anjali Tewani, Charlie Yeng, Gui Zamorano, and Grace Zhuang. Also thank you Amy Murphy and Annapaola Marconi at the Bruno Kessler Foundation for sharing with us the code of the original Kids Go Green project (https://kidsgogreen.eu/en). And thank you Darla Castelli, Sheri Burson, and the rest of the Kinetic Kidz Lab for helping make the pilot deployments a reality. Thank you Sangsu Lee for shepherding the project along after the conclusion of my senior design class, and finally, thank you Connor Fritz for joining me and agreeing to take over work on the project after my graduation, and for implementing a really cool achievement badges page.

Dylan Francis Bray

The University of Texas at Austin

December 2020

The MoveSMART Physical Activity Web Game: An Experience Report

Dylan Francis Bray, MSE
The University of Texas at Austin, 2020

Supervisor: Christine Julien

Physical activity guidelines for children recommend a minimum of 60 minutes of daily moderate to vigorous activity according to the US Department of Health and Human Services. MoveSMART is a cooperative, web-based game created to motivate increases in physical activity while integrating curriculum-based content for elementary school students. It primarily consists of a map overlaid with module waypoints that are unlocked with progress on a class's journey across the State of Texas (4th Grade) or across the United States (5th Grade), where each waypoint includes relevant curricular content. It was originally built to be used at schools in-person under the supervision of teachers. Students participate in recess or P.E. and check-in to contribute progress that advances their class through the game by swiping RFID-enabled badges and self-reporting their level of activity (less, moderately, or very active, corresponding to red, yellow, and green buttons).

Higher levels of activity correspond to farther progress in the game. However, the interaction mechanics of the game had to be changed due to the cancellation of in-person classes caused by Covid-19. To facilitate the transition to remote learning, the game was modified to enable and motivate students to participate under remote conditions.

The purpose of this work is to gather information about how to best integrate the MoveSMART game with elementary school classrooms. The report serves as a thorough description of MoveSMART, including a consolidation of existing documentation, user guides, developer guides, and other miscellaneous information about the project. It also describes the successes and lessons learned during a pilot deployment during the spring of 2020 at UT Elementary School in Austin, TX and another pilot deployment during the fall of 2020 at Hornsby-Dunlap Elementary in Del Valle ISD, Texas. In addition, it describes areas identified for future work. Each section begins by overviewing non-technical details and intended use information, describing each component's intended functionality. It then dives into technical implementation details. If your goal is to understand the system but not its implementation, ignore these technical discussions.

Contents

List of Tables

List of Figures

Chapter 1

The Game

1.1 Introduction

The MoveSMART game is a cooperative, web-based journey that motivates students to be physically active by tying real-world physical activity to in-game progress. Students check-in either online or through an RFID-enabled badge-in system to "events" that the class participates in as a group, such as P.E. and recess, or to an event corresponding to daily physical activity at home. Once an event is submitted, progress is calculated from the number of students participating, the event's duration, and each participating student's activity level (less active, active, or very active), and the class advances together on the journey. After passing waypoints along the path shown on the game map page, fun, curricular learning modules become available. In addition, the class unlocks badges after reaching certain progress and participation milestones, similar to achievements in apps like FitBit, to further motivate students. In the game, students are identified with anonymous "secret code names" to ensure their privacy. Students may track their own progress, but not that of their fellow students (without exchanging that information). Teachers are encouraged to maintain an offline list of code names corresponding to their class

rosters. This chapter gives background to this portion of the MoveSMART project, contains a non-technical user guide that explains all functionality of the site and the game, and contains a detailed developer guide with relevant technical details for developing new features or configuring a new deployment.

1.2 Background and Motivation

Physical activity is a health protective factor across ages and populations [25] and offers significant psychological and social benefits [26, 27]. For young children, physical activity is a predictor of adolescent health [21] and success in school [4]. Kids who are physically fit are more ready to learn [6], likely to attend school [1], and able to identify academic cues [7]. Because of the high coincidence of low academic performance and low levels of physical activity [3, 6, 13, 14], MoveSMART attempts a school-based intervention to increase children's awareness of and participation in physical activity. However, because prior studies show that extending physical activity interventions to include families can amplify benefits [10, 11, 24], our efforts that will build on this initial work will attempt to reach beyond the school day and school walls to promote physical activity in the wider community.

MoveSMART centers on a "serious game" that connects quantified physical activity with educational material aligned with and supplementing the school curriculum. Game-based activities within school can increase physical activity in adolescents [22], and social connectedness in a game amplifies the impact [23]. Recent studies debunk the myth that physical activity levels begin to decrease in adolescence; instead the decline begins in elementary school [8]. Few digital interventions to increase in-school physical activity have been attempted with elementary children, though preliminary work indicates that asking children to quantitatively analyze their own data improves statistical thinking skills [19, 18]. Further, prior work demonstrates that, rather than requiring dedicated time, activities to encour-

age physical activity can be integrated into the school day [20]. However, outside of data analysis, these efforts have not been tied to school curriculum.

Prior work has shown that it is possible to increase student engagement through gamification [5]. In this context, gamification uses game design elements in a non-game context to engage people and solve problems. Gamification in education is becoming more commonplace as a way to inspire learning [2]. Online learning environments facilitate social interaction [12], minimize teacher burden, and provide timely feedback [17]. The MoveSMART game applies cooperative gamification strategies and unlockable content to motivate children to achieve collective and individual goals. MoveSMART exploits an open-source gamification framework [15] that has been deployed in smart city games around Europe [16]. Notably, this framework has previously been used to implement the KidsGoGreen game [9], on which MoveSMART is directly based. Changes to the game framework required testing and pilot deployments to familiarize working with students and teachers, to validate the technical implementation, and understand key events related to implementation and deployment. This report documents technical details and key events experienced while running 2 test pilots to provide direction to better deploying the game in the future.

1.3 User Guide

This section describes the functionality of each page on the game website. Users may log in to either a teacher account that has full access or a student account that is view-only (except for check-ins). Once logged in, users must select a class. Teachers can view and interact with the entire site. They can allow students to check-in on the check-in page, view the game path and module activities on the map page, view their class and submit events on the class page, view a calendar showing events broken down by date on the calendar page, submit events for the

whole class together on the aggregate entry page, and view the badges their class has earned. Students are able to view the check-in page, the class page (graphs only), the game map page, and the badges page.

1.3.1 Logging in

The game is set up by default with both teacher and student accounts. The teacher account allows greater access, so that the teacher can manage the game in addition to experiencing all other functionality of the site. The student account is shared by every student and exposes a view-only version of the site, where students can enter their unique ID to check-in, access game modules, and view their class's progress. To log in, simply enter the credentials of the desired (teacher or student) account that were provided to you.

1.3.2 Pages

Check-In

The check-in page allows students to check-in with their level of physical activity from home. The page conveniently displays a reminder how-to guide to the student, as well as reminders about how to self-evaluate their physical activity. To check-in, a student enters his or her code name, clicks the appropriate activity level which will illuminate the traffic light, and then clicks "Submit". The results of this check-in will be reflected on the class page for the current open event. Checking in twice will overwrite the previous check-in unless a new event has been created. Check-ins are related to three different activity levels. Absences are denoted by "inactive" check-ins and are automatically populated in the absence of data. In order from least to most active, students can select:

- **Less Active:** I was not physically active (my heart was not beating fast and I was not out of breath). Denoted by red.

- **Active:** I was physically active some of the time (sometimes my heart was beating fast and sometimes I was out of breath). Denoted by yellow.
- **Very Active:** I was physically active most of the time (my heart was beating fast and I was often out of breath). Denoted by green.

Note: These check-ins must be submitted (either manually by the teacher or by auto-submit) for the progress to count in the game.

Game Map

The main content of the game is displayed on the game map page. A path marking the virtual journey is displayed with circular waypoint markers along with a location marker that displays the class' progress in the game. After the class advances past a waypoint, its learning module is unlocked and can be viewed by clicking on it. The module contents are shown with links to information about the location and various learning resources related to that location across different subjects.

Your Class

The class page shows the data entered for each event. The default view for this page is "graph view," showing bar charts of the class' activity performance. In the bottom left, teachers can click to toggle the view to the "student list view." The student list view shows each student's individual check-in response for every previous event. In addition, this page can be used as another way for students to check-in, for teachers to submit check-ins for students, or as a way for teachers to modify check-ins before submitting them. To select activity levels, click the artist's palette icon button in the bottom right, then select the desired activity level. Clicking the box for each student in the column of the open event updates their activity level. After the completion of a class activity, the event should be submitted. If this is not done,

the next time the badge system is used, it will overwrite data. To submit data, fill in the event title, enter the date, click a weather icon, then select an approximate duration. Then, click "Send Data." It may prompt you with a warning that any remaining students that did not badge in will be recorded as absent. Also note that events cannot be reopened, so pay attention when closing events!

Calendar

The calendar page displays data from events listed on a calendar. It can display data in two different layouts, as well as display graphs containing different information. The idea is to allow students to understand how their progress through the game is achieved, building numerical intuition and understanding data presentation. Clicking on the event loads a bar chart with data about the component swipes of an event. There is also a "switch view" button that toggles the view to a calendar that displays the distance traveled by type of check-in (red, yellow, green) on each day, as well as appropriately modified charts.

Aggregate Activity Entry

The aggregate activity page is useful for adding events if you know everyone participated in an activity but do not wish to spend the time having every student check-in. It is also useful if you want to add progress to the game and unlock a module slightly before the class actually reaches and unlocks that module. This page also shows every event your class has previously checked in for in a convenient card-based format.

Settings

The settings page allows the teacher to enable, disable, and add students as well as enable or disable midnight auto-submit on the settings page. Each class/team is

pre-populated with 25 students, but some are disabled because no class currently contains 25 or more students. In the event a student leaves the class, the teacher may select the student and deactivate them. Then, the student's previous activity will be hidden but remain counted towards progress in the game. If the same student returns, they may be re-enabled, preserving previous check-in activity. In the event a new student joins the class, an inactive student may be activated. If this is a student who participated in the game previously, it is recommended to select that student and re-enable them. If the student is new to the class, the teacher should enable one of the previously unused, pre-populated students. If these pre-populated students been exhausted, then the teacher may add a student manually by assigning a new code name.

Activity Management

This page is under development but will allow teachers to modify waypoint module content and define their own activities.

Badges

The badges page displays the two types of badges that can be unlocked by teams as a motivational tool, similar to those seen in popular fitness tracking apps like FitBit. Teamwork badges are unlocked through class-wide participation streaks, while distance badges are unlocked when the class travels past distance milestones. Badges are associated with third party articles to expose students to new topics, relate their activity to the real world, and build their intuition about numbers and distances.

1.4 Developer Guide

MoveSMART is a single-page web application built using the AngularJS framework (https://angularjs.org/) for the front-end and Spring Boot 1.5.8 (https://docs.spring.io/spring-boot/docs/1.5.8.RELEASE/reference/html/) for the back-end, which is backed by a MongoDB (https://www.mongodb.com/) database. This document overviews the application structure, provides code documentation, and explains some rationale behind design choices. Prior to reviewing this section, it is important to understand the motivation behind this project and the existing functionality. If you are not already familiar, you are strongly encouraged to review the User Guide as well as set up a local instance to play around with. This section first describes how to get started developing and deploying MoveSMART and the provides more detailed documentation about different aspects of the codebase.

1.4.1 Local Development Setup

This section describes the required software tools and how to set up the game so that it runs locally from the IntelliJ IDE.

1. Download and install MongoDB: https://www.mongodb.com/download-center/community
2. Download and install IntelliJ: https://www.jetbrains.com/idea/
3. Clone the git repository to your local machine.
4. Open the project and import as Maven project.
5. Inspect `application.properties` to set the value of `spring.data.mongodb.host` to `localhost/<connection_string_details>` (include relevant connection string details). The file can be found in the following path: `sco.climb.game-dashboard/src/main/resources/application.properties`

6. Open a terminal and navigate to a directory where you would like to keep your local MongoDB data.

7. Type `$ mongod --dbpath=.` to run MongoDB locally.

8. In IntelliJ, find `sco.climb.game-dashboard/src/main/java/ClimbDashboardApp.java`, then right click it and select either run or debug.

9. Open a web browser and navigate to http://localhost:5000.

1.4.2 Version Control

The main Git branch and branches corresponding to deployed versions of the game contain configurations that are different from those used in development and testing branches. The file `sco.climb.game-dashboard/src/main/resources/application.properties` contains several such fields. The fields `spring.data.mongodb.host`, `db.name`, and `server.port` should be different between each branch. In addition, the analytics `gtag` configuration found in `sco.climb.game-dashboard/src/main/resources/static/index.html` should only be set in deploy branches. Because they are different, changes to main or develop should be performed on top of changes merged in from development branches. **Never make development changes on the main or any deploy branches** (except for specific configuration changes). Only merge changes from other branches into main. Main may be merged into deploy but deploy should never be merged into main. The goal is to prevent configurations from getting crossed, scrambled, or lost.

1.4.3 Deployment

The main steps behind deployment to AWS are generating `game-dashboard.jar`, zipping it together with the folder `.ebextensions`, and uploading the resulting `.zip` file to AWS. Follow the step-by-step guide below for deployment instructions:

Generating Deployable Zip Archive

1. **IntelliJ Only** Make sure "Delegate IDE build/run actions to Maven" is checked. This can be found by clicking File > Settings > Build, Execution, Deployment > Build Tools > Maven > Runner.

2. Ensure that `spring.data.mongodb.host` and `db.name` are correct in the `sco.climb.game-dashboard/src/main/resources/application.properties` file. Also ensure that `server.port` is set to `5000`, which is the default port for AWS deployments.

3. Build the project in IntelliJ or run `mvn package` in the top level project directory. This creates `game-dashboard.jar` in the `sco.climb.game-dashboard/target` directory.

4. Zip `game-dashboard.jar` and the `.ebextensions` directory together.

Creating a MongoDB Instance

1. Create a MongoDB account: (https://www.mongodb.com/)

2. You should be automatically redirected to create a new cluster.

3. Select the default (and free) options and click "Create Cluster".

4. Go to the security tab and you'll arrive at MongoDB Users

5. Create one user with a secure username and password with User Privileges "Read and write to any database".

6. Create another user with a secure username and password, and User Privileges "Atlas admin".

7. Navigate to the "IP Whitelist" tab. If you know what you are doing, you can set this so the database is only accessible to you and to your Amazon instance.

For easy but less secure setup, add a white list entry "0.0.0.0/0" and comment "everybody". This allows any IP address on the internet to connect to the database. Still, only those with credentials can log in.

8. Go back to the overview pane and once the database is ready go to Connect > Connect Your Application and copy the string.

9. Within the software files for the project, change the value of `spring.data.mongodb.host` in the file `sco.climb.game-dashboard/src/main/resources/application.properties`. This will allow your program, running locally or deployed, to connect to this MongoDB database. Be very careful that you do not deploy with a connection to the wrong database or you do not run the development server with a connection to the wrong database. This could interfere and damage an existing deployment's data.

Configuring and Uploading to AWS

Note: This section assumes you have never used AWS before. You may have to set up a compute instance; a t2.micro instance is sufficient.

1. Navigate to the AWS Console (https://aws.amazon.com/console/) and sign in.

2. In the search bar, type "Elastic Beanstalk" and select it.

3. Near the top right click "Create a new environment".

4. Select the "Web server environment".

5. Fill out the form choosing an appropriate name, domain, and description.

6. Select "Managed platform", "Java" under platform, and "Java 8 running on 64bit Amazon Linux" under platform branch. Use the default version.

7. Select "Upload your code", enter your desired version label, select "Local file" and upload the `.zip` file you created in the subsection above.

8. Click "Create environment".

Updating an Existing AWS Instance

1. Navigate to the AWS Console (https://aws.amazon.com/console/) and sign in.

2. In the search bar, type "Elastic Beanstalk" and select it.

3. Select the instance you wish to update.

4. Click "Upload and deploy".

5. Choose the `.zip` file created earlier in this section and name the version something descriptive.

6. Click "Deploy"

1.4.4 Front-End

The front end is built using AngularJS and attempts to adhere to the Spring MVC - AngularJS design pattern explained in this blog post: https://blog.angular-university.io/developing-a-modern-java-8-web-app-with-spring-mvc-and-angularjs/. An AngularJS application is composed of views, controllers, and front-tend services.

- Views are the HTML templates that the user sees and can be found in `sco.climb.game-dashboard/src/main/resources/static/templates`.

- Controllers are JavaScript files that control the viewable content of each web page. The current implementation does not always abide by this; many controllers contain functionality that should be left for the service. Where possible, err towards refactoring so that controllers only contain code that directly manipulates the view templates. Controllers can be found in `sco.climb.game-dashboard/src/main/resources/static/js/controllers`.

- Services are JavaScript files that provide helper functions and the business logic of the front end. The current implementation does not always abide by this; many services do not contain much business logic since it is all left to the controller. Where possible, err towards refactoring so that services contain all code that does not directly manipulates the view templates. Services can be found in `sco.climb.game-dashboard/src/main/resources/static/js/controllers`.

Every page has an HTML template, a service, and a controller named something self-explanatory and related. When working on the front end, be sure to follow the design pattern recommended above. The rest of this section contains useful information about remaining parts of the front end.

Login

Logging in to the project is handled, as you might expect, through the login service in `loginSrv.js`. The login functionality works by storing certain values in the browser's `localStorage`. This allows the user session and associated metadata to persist while the browser session is opened. All pages getting the currently selected game, class, or user role should go through `loginSrv.js`.

Navigation

There is a navigation bar that can be edited in `sco.climb.game-dashboard/src/main/resources/static/templates/home.html`. However, it is also necessary to register pages with the Angular `stateProvider` in `app.js` for them to be reachable in the single page application.

Adding a Page

In order to add a page, you must follow several steps. You must register the service and the controller in `sco.climb.game-dashboard/src/main/resources/static/index.html` and in the router found at `sco.climb.game-dashboard/src/main/resources/static/js/app.js`. You must also include the template HTML in `app.js`. Then you must modify the navigation bar in `sco.climb.game-dashboard/src/main/resources/static/templates/home.html` for the page to be reachable. You must also register the services that each service and controller depend on explicitly when creating the service and controller. See existing code for an example of this.

Analytics

An anonymous user tracking instance was set up using Google Analytics events that are parsed to create sequences of interactions from single users ('interaction timelines') and summary statistics. A user session ID is created and tracks events through each user session, creating an interaction timeline. Page views and some button clicks are monitored. The data downloaded from Google Analytics can be parsed by scripts found in the `/analytics` directory of the project. It is important to note that test deployments do not report tracking to Google Analytics. The `gtag` configuration in `sco.climb.game-dashboard/src/main/resources/static/index.html` is undefined for all non-deployed branches. It is important to ensure it remains unset for development and testing and remains set for active deployments.

See the discussion of version control with GitHub above for more information.

1.4.5 Back-End

As mentioned in the front-end section, the back-end is built using Java Spring Framework and attempts to adhere to the Spring MVC - AngularJS design pattern explained in this blog post: https://blog.angular-university.io/developing-a-modern-java-8-web-app-with-spring-mvc-and-angularjs/. The code is broken up into 2 layers, the router layer and the persistence layer. Most Java code referenced below can be found in `sco.climb.game-dashboard/src/main/java/it/smartcommunitylab/climb/gamification/dashboard`. Also note that `ClimbDashboardApp` is the main entrypoint to the application.

- The router layer defines controllers. These are the files that map URLs to functionality and handle all requests from the front-end. The controllers are separated by area of concern (generally by front-end page) and are:
  - `CalendarController.java`
  - `EventController.java`
  - `GameController.java`
  - `PlayerController.java`
  - `TeamController.java`
- The persistence layer defines all database access functions. `RepositoryManager.java` contains all database accesses.

Configuration

The database is configured by reading pre-defined school, game, team, and player data from `appConfig.json` into `AppConfig.java`. The file `sco.climb.game-dashboard/`

`src/main/resources/application.properties` contains most important configuration properties. The fields `spring.data.mongodb.host` and `db.name` should be different between each deployment and when testing, because they correspond to actual databases that could be live. The value of `server.port` should be set to `5000` for AWS deployment. In addition, the analytics `gtag` configuration found in `sco.climb.game-dashboard/src/main/resources/static/index.html` and should only be set in deploy branches. The game path is defined piece-wise in `appConfig.json` and is composed of specially encoded strings called Polylines. These must be generated by a specific API as follows: Do a GET request with the following URL (all as one contiguous URL): `https://maps.googleapis.com/maps/api/directions/json?Origin=<previous>&destination=<current>&key=<APIkey>` where origin and destination are (latitude, longitude) pairs as follows: `Origin="231.7619,-106.4850"`. Occasionally, the result of this query will fail to display due to containing illegal characters inserted when copying from a browser. Remove these characters (usually backslashes) to fix the string.

Controllers

The controllers in `/controller` define all API routes in the game. The controllers are broken up so as to each correspond to a specific type of action that mostly interacts with a single page of the website. For example, `EventController.java` provides API routes for creating, reading, saving, and submitting events at the URL `/api/events/*`.

Converters

These converters are used to interpret `ZonedDateTime` objects serialized and saved to the database. The default to Central Time.

Security

Because the site collects anonymous, non personally identifiable data, API routes are currently open and in plaintext. Users log in with `token` objects that include user roles and game information, but this token is not curently applied to the API. Spring framework is well integrated with a security tool called Spring Security, which can serve as a layer in front of every API route, enforcing a valid token is used. Implementing this is an important step that should be completed before a wider deployment with any potential identifying data stored on the site.

Storage

The Java Spring back-end interacts with the MongoDB instance through `/storage/RepositoryManager.java`. All MongoDB queries are found in this file. Note that you should not write database queries from any of the controller files.

1.4.6 Database

The database is a remotely hosted MongoDB instance and is accessed using Spring Data MongoDB: https://spring.io/projects/spring-data-mongodb. Calls to the database are all handled in `RepositoryManager.java` and most configuration is set in `application.properties`.

Data Model

Every object stored in the database extends `BaseObject`, which defines useful reference fields. It defines `_id` which is used by MongoDB to define its own ID (do not overwrite this field). MongoDB ensures uniqueness of this field. The field `friendlyId` is an ID that can be set by the user, which is useful when performing manual configuration of the site. It also contains fields identifying the creation date and the latest update. Objects are stored in a "reverse hierarchy" where objects

lower in the tree have references to the associated objects higher in the tree. This helps with MongoDB queries, since instead of querying an object and then querying a list of its children, we can simply query the objects that have `parentId`, which returns a collection much faster. Figure 1.1 shows the relationship between every object that is stored in the database.

Resetting

Upon restarting the game website, the database is **not** overwritten by default. The default behavior is that, once a game is created, only the users may change the data. If something goes wrong, resetting the database can be done through the MongoDB Atlas web interface or MongoDB Compass local interface by deleting the collection or the object in the collection. If upon startup, the game realizes a required object is missing, a default instance will be created and persisted.

1.4.7 Typical Data Flow

A typical HTTP call from front-end to back-end has 4-5 steps. It is helpful to trace these calls to understand the application's data flow when getting started with MoveSMART. You should also familiarize yourself with JavaScript promises.

1. The call from the front-end will occur in a controller (e.g. `classCtrl.js`).
2. The controller will make a call using a service (e.g. `classSrv.js`).
3. The service will make a call through the data service file through which all HTTP calls pass (`dataSrv.js`).
4. A REST API endpoint in the back-end is hit in e.g. `DashboardController.java` based on the URL used in the HTTP call.
5. The MongoDB database is updated or searched in e.g. `RepositoryManager.java` through the e.g. `RepositoryManager` object in e.g. `DashboardController.`

`java` named `storage`. A value is possibly returned all the way back to the front-end.

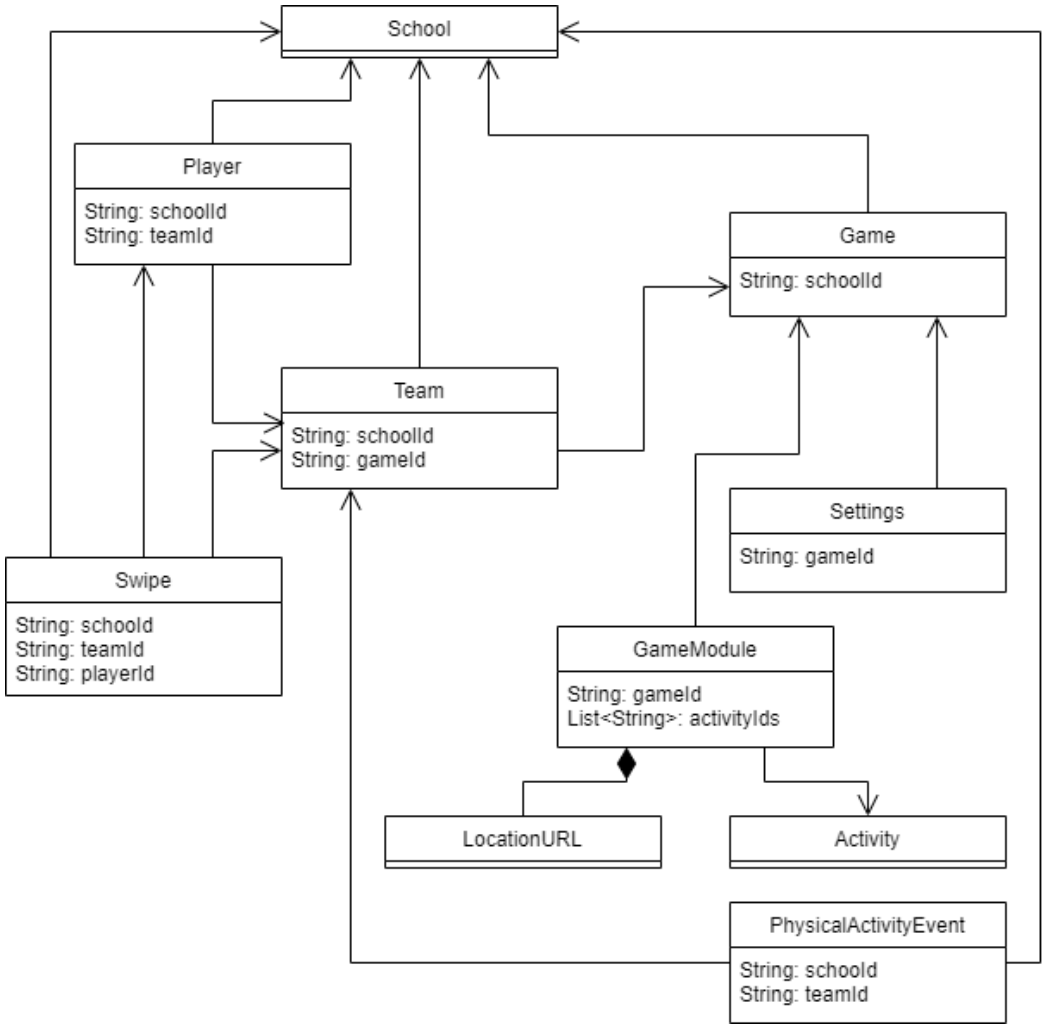

Figure 1.1: Data Model Relationship

Chapter 2

RFID Check-In Box

In addition to using the check-in web page, a student can check in by scanning an RFID badge, then selecting his or her perceived level of activity by pushing a colored, LED-enabled button, which delivers visual feedback of the student's selection by flashing the button's LED. Each box is designed to be powered by a Raspberry Pi 3b+ that controls an RFID sensor and 4 LED-enabled buttons, and is constructed of 0.25 inch thick acrylic pieces, with button and RFID scanner mounts cut into the top and a charging cable opening cut into the back. A box measures 6.5 in x 4.25 in x 3.75 in. This section includes a user guide that describes how to power on and use the box, a quick-start developer guide that explains how to reconfigure an existing box for a new deployment, a Raspberry Pi setup guide that overviews recommended configuration settings, and a construction guide.

2.1 User Guide

Using the RFID check-in box is easy. To power on, simply plug in the power cable to an outlet and, if necessary, click the switch on the power cable. The yellow button should illuminate while the box is starting up. Wait until the green button begins

flashing, then press it. Now the box is ready to go. Swipe an RFID card, which should trigger a beep and cause all buttons to light up, then select the desired input. The selected button should flash a few times and all buttons should turn off. Now the box is ready for the next input. Check-ins are related to three different activity levels, while absences are denoted by "inactive" check-ins and are automatically populated in the absence of data. In order from least to most active, students can select:

- **Less Active:** I was not physically active (my heart was not beating fast and I was not out of breath). Denoted by red.
- **Active:** I was physically active some of the time (sometimes my heart was beating fast and sometimes I was out of breath). Denoted by yellow.
- **Very Active:** I was physically active most of the time (my heart was beating fast and I was often out of breath). Denoted by green.

You can view the check-ins by refreshing the "Your Class" page of the web game. Checking in twice will overwrite the previous check-in unless a new event has been created.

Note: These check-ins must be submitted (either manually or by auto-submit) for the progress to count in the game.

2.2 Developer Guide

2.2.1 Configuration Quick Start

Note: This quick start guide assumes a previously configured RFID box.

There are only two files that are important to a deployment. The first is `Hardware.py`, which is the Python 2 script used to take input from button presses, control the

Figure 2.1: A fully constructed RFID Check-In Box

button LEDs, and read inputs from the RFID reader. The second is `rc.local`, which is used to tell the Raspberry Pi to run `Hardware.py` on startup. Perform the following steps for configuration:

1. Modify the variable `domain` in `Hardware.py` to specify the desired game URL. There is no required location for this file, but it is often found in `/home/pi/Documents/project_smart`.

2. Delete `Hardware.log` so that the new log doesn't contain a log of old swipe data. It is found in the same directory as `Hardware.py`.

3. Inspect, and modify if necessary, `/etc/rc.local` so that it reflects the correct path to `Hardware.py`. If desired, `rc.local` can also be modified to enable ssh on startup. See the copy of `rc.local` in the code repository for details on how to do this.

4. Test the box by rebooting it and swiping some cards. This step will only work if the database is configured correctly with player `cardRfid` fields corresponding to the RFID cards you are using.

5. If there are any errors, inspect `Hardware.log`, which will be created in the same directory as `Hardware.py`.

2.2.2 Raspberry Pi Setup

This section assumes a blank SD card or a Raspberry Pi that you wish to erase and re-image. If you do not wish to erase and re-image your existing Raspberry Pi, skip to step 5. The instructions also assume you install Raspberry Pi OS; your setup options may vary if you choose to install a different OS. Note that any Raspberry Pi should work, but differences in USB connections and internet connectivity are present across Raspberry Pi devices that may require different hardware and deviating from instructions in this document. To set up a new Raspberry Pi, follow the steps below:

1. Download and install the Raspberry Pi Imager or follow any of the options to manually install an operating system image from https://www.raspberrypi.org/software/.

2. Insert the SD card into your computer's SD card reader.

3. Open Raspberry Pi Imager, select "Raspberry Pi OS (32-bit)" and the SD card, then click "Write". Alternatively, follow whichever instructions you wish to use. Wait for this to complete.

4. Once the SD card has been successfully written, remove it from your computer and insert it into the Raspberry Pi.

5. Connect the Raspberry Pi to a monitor, a keyboard, and a mouse.

6. Select your desired settings as prompted (e.g. US, American English, Chicago time zone, US Keyboard).

7. Find the files `Hardware.py` and `rc.local` in the `/rfid-box` directory of the project code repository.

8. Download `Hardware.py`. There is no required location for this file, but it is often saved to `/home/pi/Documents/project_smart`.

9. Modify `/etc/rc.local` so that it includes the startup script and reflects the correct path to `Hardware.py`. If desired, `rc.local` can also be modified to enable ssh on startup. See the copy of `rc.local` in the code repository for details on how to do this.

10. Test the box by rebooting it and swiping some cards. This step will only work if the database is configured correctly with player `cardRfid` fields corresponding to the RFID cards you are using.

11. If there are any errors, inspect `Hardware.log`, which will be created in the same directory as `Hardware.py`.

2.2.3 Box Construction

A MoveSMART RFID Box is made of 6, 0.25 inch thick, laser cut acrylic pieces. Fully constructed, it measures 6.5 in x 4.25 in x 3.75 in. A piece with holes cut out for buttons and the RFID scanner mount is the top of the box. The row of three buttons should be farther away from the user, with the RFID mount being closer. The piece with the cable opening is the back side. Regular Gorilla glue or acrylic adhesive can be used to assemble the box, but avoid using super glue. Designs files can be found in the project repository. Components are wired together according to 2.2; however, if you have an assembled box, it may be easier to simply copy the

existing wiring rather than reading the schematic. See table 2.1 for a reference of components required for assembly of the box.

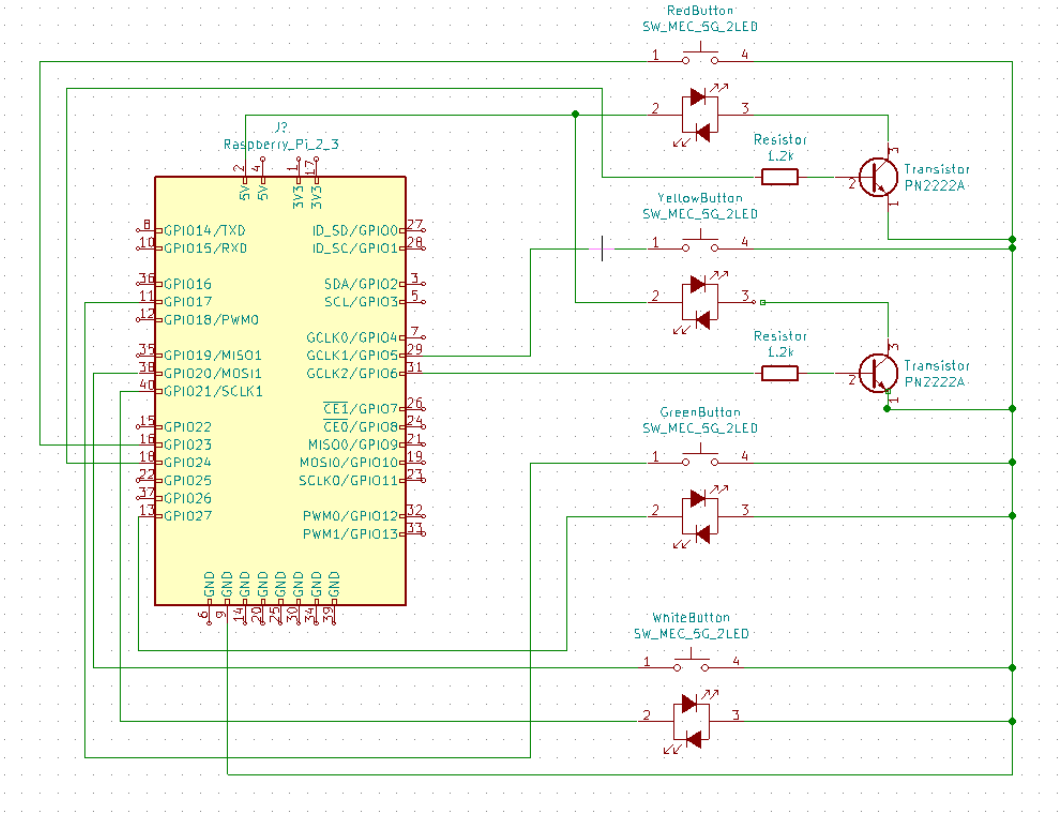

Figure 2.2: RFID Check-In Box Wiring Schematic

Item	Quantity	Link
Acrylic box set	1	Custom laser cut; see drawings
Raspberry Pi 3b+	1	https://www.adafruit.com/product/3775
Raspberry Pi 3b+ case	1	https://www.adafruit.com/product/2258
Raspberry Pi power supply	1	https://www.adafruit.com/product/1995
SD card	1	https://www.sparkfun.com/products/14832
RFID kit	1	https://www.sparkfun.com/products/13198
USB Mini-B cable	1	https://www.sparkfun.com/products/13243
4-40 Pan Head Machine Screw Phillips Drive	4	https://www.digikey.com/en/products/detail/keystone-electronics/9901/317322
4-40; 3/8in standoffs	4	https://www.sparkfun.com/products/10461
White button	1	https://www.digikey.com/en/products/detail/adafruit-industries-llc/3429/7349491
Red button	1	https://www.digikey.com/en/products/detail/adafruit-industries-llc/3430/7349492
Yellow button	1	https://www.digikey.com/en/products/detail/adafruit-industries-llc/3431/7349493
Green button	1	https://www.digikey.com/en/products/detail/adafruit-industries-llc/3433/8119804
Breadboard	1	https://www.adafruit.com/product/64
Jumper wires M-M	~ 20	https://www.sparkfun.com/products/12795
Jumper wires M-F	~ 20	https://www.sparkfun.com/products/12794
Wires for buttons	4	https://www.adafruit.com/product/1152
1.2 kΩ axial resistor	2	https://www.digikey.com/en/products/detail/stackpole-electronics-inc/CF14JT1K20/1830352
PN2222A transistor	2	https://www.digikey.com/en/products/detail/on-semiconductor/PN2222ATA/3042489
RFID cards	Optional	https://www.sparkfun.com/products/14325

Table 2.1: RFID Check-In Box Required Materials

Chapter 3

Pilot Deployment

3.1 Introduction

MoveSMART was deployed in two pilots during 2020. The first, at UT Elementary School in Austin, TX, ran from January to March and was interrupted by the Covid-19 pandemic and associated school closures. It served primarily to test the data collection from the RFID Box in a P.E. class setting. In addition, a team of kinesiology researchers collected accelerometry data to determine how accurately students are able to self-report their physical activity. That work is ongoing. The second, at Hornsby-Dunlap Elementary School in Del Valle ISD, near Austin, TX, began in mid-October and ran through the conclusion of the semester in December. It served to continue the efforts related to understanding student self-reporting of activity, to validate changes made to the functionality of the game in response to Covid-19 and the ensuing virtual learning environment, and also as a means to gain experience deploying the system in a limited environment prior to a wider deployment. The following sections include information about the experience of running each pilot and various lessons learned that are applicable to future work.

Functionality, interactions with teachers, interactions with students, and in-

tegration with curriculum are important aspects to understand in the context of an elementary classroom. The pilot deployments helped validate some of these key aspects, while other aspects were not thoroughly validated. The game's basic functionality was successfully implemented and used by classes at the school. It performed as expected with few bugs; user interface issues were the biggest obstacle teachers faced. In addition, changes made to accommodate for remote learning due to Covid-19 were mostly successful. However, motivating students requires additional work. Unlockable waypoints serve this purpose in the game and a new unlockable milestone badge system was released during deployment, however, these features were not well used due to slow progress during the game. Overall, teachers remained engaged and expressed interest in certain changes, including new user interface features, printable content for classroom bulletin boards, and direct support from the research team, providing a solid foundation of lessons learned and areas for improvement.

3.2 UT Elementary Pilot Deployment

The deployment at UT Elementary School consisted of one 4th grade and one 5th grade class and was primarily focused on P.E. Students in each class participated in each day's activity, then reported their activity through check-ins with the RFID-box, sometimes under close teacher supervision, and sometimes under less strict supervision. During observation of the class, it is important to note that the teacher often, but not consistently, reminded students to remember their activity across the whole class and described each activity level using terms like "out of breath," to describe the "very active" level of activity. We attempted to replicate this type of reminder on the check-in page created for the Covid-19 virtual learning deployment described in the next section. It appeared that these reminders may have encouraged students to report less activity. It is not certain, but quite possible, that this

reporting was was more accurate due to increased awareness caused by these reminders. In addition during sessions with extra observation from the researchers, students appeared to report less activity. It is not clear whether this is the result of students being more honest in their reporting, or whether they perceived a need to under-report due to closer supervision.

Students participating in this deployment heavily used the box, but did not interact with the game interface, due to Covid-19 school closures that occurred during the middle of the pilot. Students were able to check-in using RFID cards in a reasonable amount of time at the conclusion of PE; however, it did contribute some overhead to the class because the cards were stored in a wall rack and not kept with students at all times, so the students had to spend time finding their cards. Integrating with existing school ID cards or developing an even faster logistical solution to this problem would make the check-in system less obtrusive for the class. During some observation sessions, the researchers assisted with distributing cards during the cool-down activity. This may be a helpful way of integrating the check-in box with PE classes less obtrusively.

3.3 Hornsby-Dunlap Elementary Pilot Deployment

The deployment at Hornsby-Dunlap Elementary School consisted of four 4th grade and four 5th grade classes. A PE teacher served as a liaison between the MoveSMART team and the grade level teachers. To facilitate the game, the PE teacher encouraged her students to participate and tracked each class' activity during virtual, and later in person, PE by submitting events with the aggregate activity entry feature. However, it quickly became clear that individual teachers are crucial to achieving strong participation. Some classes experienced very few, and eventually no daily check-ins, while other classes consistently experienced up to half participation.

3.3.1 Covid-19 Changes

In order to accommodate students learning from home, we had to develop a "student view" of the website that restricts access to game management features and other students' data, in addition to creating a page that replaces the RFID Box and allowed students to check in online. Because the game website did not support user accounts in any elegant way, the student account is shared between all students and allows the user to log in the same was as a teacher, but includes a flag that causes pages to hide elements that are reserved for teachers only, such as viewing the class check-in view and editing settings. Because students share the same account, we originally envisioned that students would use their "secret codename" aliases to check in, but this proved difficult, since many of these aliases are long, difficult-to-spell words. We quickly had to revert to using student ID numbers, despite fearing that students could interfere with each other's check-ins. However, this issue did not occur during this deployment (which is not to say it never would). This issue can be resolved by implementing more formalized account management that integrates with school district single sign on technology.

It was quickly evident that, without consistent teacher supervision and intervention, many students would not use the game. To address this, in addition to creating the student view, we implemented an achievement badge system, similar to what exists in modern fitness apps like FitBit, that unlocks badges at certain progress milestones as well as class participation milestones. The badges are associated with third party articles to expose students to new topics, relate their activity to the real world, and build their intuition about numbers and distances.

Because students were given access to the site from home, without teacher supervision, we also wanted to understand how students interacted with the game. An anonymous user tracking instance was set up using Google Analytics events that are parsed to create sequences of interactions from single users (‘interaction

timelines') and summary statistics. This will help us learn what parts of the site receive more use and what parts are ignored, so we can tailor our efforts accordingly.

In summary, the following changes and additions were made to accommodate virtual learning:

- Student log in to the game and related access restrictions
- Online check-in page
- Automatic daily event submission
- Progress and participation badges
- Google Analytics

3.3.2 Lessons Learned

Many of these changes were successful; however, we still struggled with student and teacher participation. Some classes had much higher participation rates, with half of the class checking in every day on their own, while others had few or no check-ins every day. This section describes ways to better integrate the game with classes and how to capture student excitement, thoughts on how to design more effective user interfaces, and ways to increase support of the teachers.

Teachers appear to be interested but unsure of how to effectively use the game in their classrooms. To this end, it would be helpful to develop more comprehensive materials to integrate the game into the classroom, and to provide teachers with more focused guidance about how to use the game in their classrooms. Teachers also reported that students are excited about the game and reiterated a need to capture that excitement. They suggested creating printable content that could be posted to hallway bulletin boards to give a passive visual reminder to students while

on campus. Some ideas for this content include a poster of the traffic light visual included on the check in page as a reminder of the red, yellow, and green physical activity levels. Another idea included displaying the most recently earned badges for each class, In addition, teachers suggested that promoting healthy competition between classrooms would be a useful way of keeping students engaged and more motivated to consistently report their progress. Modifying the web-interface to show a shoe icon or something similar that indicates progress relative to other classes could accomplish this. A great participation success one teacher had was by giving students time directly after physical activity to check-in, which is an easy way to get students into the habit of using the game.

Another important consideration is that the website should be impossible to break, and it should give instant feedback about what's wrong with the user's attempted actions. We suffered a delay due to this; the site was not broken, but the teacher didn't understand how to submit data that was the same size of the class. Do not assume the user will dig through a detailed guide! To address this, a dialog now pops up and provides feedback for the exact reason an action is rejected (too few or too many students reported). Relatedly, it is important to provide inputs that are as forgiving as possible. If something is missing, accept the data and provide a default case. This dialog was updated to infer that if too few students were entered, assume the remainder are inactive. Only reject inputs when it is impossible to infer a useful action being taken. In addition, some students experienced issues with logging in. This is not acceptable and a permanent solution is necessary. Integrating with school single-sign-on services could accomplish this.

Teachers also expressed a desire for a consistent point of direct contact. A centralized teacher contact was originally used for this pilot, but teachers wanted someone they could reach out to directly. A live demo tutorial to the class as well as brief, weekly check-in video calls to promote the game and provide updates and

reminders would also be useful.

Finally, the speed at which classes advanced through the game was too slow and needs additional calibration. Part of this likely has to do with the low participation, but even among the better performing classes, they only unlocked two modules over the course of eight weeks, slower than the every-two-weeks pace we had targeted. Getting this right was not a primary concern during this pilot, but is important to note for future deployments.

In summary, the following actions would make the game and deployment experience better:

- Create printable content for bulletin board displays.
- Encourage healthy competition by showing progress for every class on class selection page.
- More prominent badges functionality on the site or in the game (e.g. latest badge earned)!
- Encourage teachers to provide check-in time directly following activity.
- Design user-centric, forgiving site elements.
- Integrate with single-sign-on service.
- Provide centralized point of contact.
- Actively "pop in" to show tutorials and give updates.
- Calibrate advancement speed and module placement.

3.4 Implications for Children's Physical Activity Work

As mentioned throughout this report, several key factors, including communication with teachers and student motivation are crucial to success. Often, classroom changes require a champion who can effectively advocate to teachers and administration, and P.E. teachers are a great resource for this. However, establishing direct contact with teachers is important to provide a line of communication for feedback from the classroom. Using the P.E. teacher as a liaison between researchers and other teachers was overwhelming for the P.E. teacher, confusing for the other teachers, and otherwise inefficient. Active student participation is also key, but it is difficult to achieve this without direct teacher intervention that encourages participation. Providing small amounts of time for reflection and reporting during the school day was the number one factor that increased class participation. Students were excited about the game, suggesting that the motivation features are working to a degree, but did not self-direct that excitement into participation. Finally, more consistent, direct contact with teachers and students in the classroom is important. Teachers were not well-versed enough with the game to provide demos to their classes or to understand everything they could do immediately, so they requested short demos from the research team as well as short, regularly scheduled check-ins with each class while getting started. Overall the game shows much promise, especially with the implementation of suggested changes discussed earlier.

Bibliography

[1] E. E. Centeio, Jessica Duncan Cance, Jeanne M Barcelona, and Darla M Castelli. Relationship between health risk and school attendance among adolescents. *American J. of Health Education*, 49(1):28–32, 2018.

[2] C. Cheong and J. Filippou. Quick Quiz: A Gamified Approach for Enhancing Learning. In *PACIS*, 2013.

[3] D. P. Coe, Thomas Peterson, Cheryl Blair, Mary C Schutten, and Heather Peddie. Physical fitness, academic achievement, and socioeconomic status in school-aged youth. *J. of School Health*, 83(7):500–507, 2013.

[4] Jonathan M. Cosgrove, Yen T. Chen, and Darla M. Castelli. Physical fitness, grit, school attendance, and academic performance among adolescents. *BioMed Research International*, 2018.

[5] A. Dominguez et al. Gamifying learning experiences: Practical implications and outcomes. *Computers & Education*, 63:380–392, April 2013.

[6] J. E. Donnelly et al. Physical activity, fitness, cognitive function, and academic achievement in children: a systematic review. *Medicine and Science in Sports and Exercise*, 48(6):1197, 2016.

[7] E. S. Drollette et al. Effects of the FITKids physical activity randomized controlled trial on conflict monitoring in youth. *Psychophysiology*, 55(3), 2018.

[8] M. A. Farooq et al. Timing of the decline in physical activity in childhood and adolescence: Gateshead Millennium Cohort Study. *Br J Sports Med*, 52(15):1002–1006, 2018.

[9] Matteo Gerosa, Annapaola Marconi, Marco Pistore, and Paolo Traverso. An Open Platform for Children's Independent Mobility. In *Smart Cities, Green Technologies, and Intelligent Transport Systems*, Communications in Computer and Information Science, pages 50–71, 2015.

[10] J. M. Guagliano et al. The development and feasibility of a randomised family-based physical activity promotion intervention: the Families Reporting Every Step to Health (FRESH) study. *Pilot and Feasibility Studies*, 5(1), February 2019.

[11] A. S. Ha et al. Promoting physical activity in children through family-based intervention: protocol of the "Active 1+FUN" randomized controlled trial. *BMC Public Health*, 19(1), February 2019.

[12] L. Hakulinen and T. Auvinen. The Effect of Gamification on Students with Different Achievement Goal Orientations. In *2014 International Conf. on Teaching and Learning in Computing and Engineering*, pages 9–16, April 2014.

[13] M. L. Humbert et al. Factors that influence physical activity participation among high-and low-SES youth. *Qualitative Health Research*, 16(4):467–483, 2006.

[14] I. Janssen and A. G. LeBlanc. Systematic review of the health benefits of physical activity and fitness in school-aged children and youth. *International J. of Behavioral Nutrition and Physical Activity*, 7(1):40, 2010.

[15] R. Kazhamiakin, A. Marconi, A. Martinelli, M. Pistore, and G. Valetto. A

gamification framework for the long-term engagement of smart citizens. In *IEEE International Smart Cities Conf.*, pages 1–7, 9 2016.

[16] R. Khoshkangini, G. Valetto, and A. Marconi. Generating personalized challenges to enhance the persuasive power of gamification. In *International Workshop on Personalizing Persuasive Tech.*, 2017.

[17] P. A. Kirschner and A. C. Karpinski. Facebook and academic performance. *Computers in Human Behavior*, 26(6):1237–1245, November 2010.

[18] V. R. Lee and J. Drake. Quantified recess: design of an activity for elementary students involving analyses of their own movement data. In *Proc. of the 12th International Conf. on Interaction Design and Children*, pages 273–276. ACM, 2013.

[19] V. R. Lee, J. R. Drake, and J. L. Thayne. Appropriating quantified self technologies to support elementary statistical teaching and learning. *IEEE Trans. on Learning Tech.*, 9(4):354–365, 2016.

[20] V. R. Lee, Joel R Drake, Ryan Cain, and Jeffrey Thayne. Opportunistic uses of the traditional school day through student examination of Fitbit activity tracker data. In *Proc. of the 14th International Conf. on Interaction Design and Children*, pages 209–218. ACM, 2015.

[21] A. Lukacs, P. Sasvari, and E. Kiss-Toth. Physical activity and physical fitness as protective factors of adolescent health. *International J. of Adolescent Medicine and Health*, 2018.

[22] A. Macvean and J. Robertson. iFitQuest: a school based study of a mobile location-aware exergame for adolescents. In *Proc. of the 14th International Conf. on Human-Computer Interaction with Mobile Devices and Services*, pages 359–368. ACM, 2012.

[23] A. D. Miller and E. D. Mynatt. Stepstream: a school-based pervasive social fitness system for everyday adolescent health. In *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, pages 2823–2832. ACM, 2014.

[24] P. J. Morgan et al. Engaging Fathers to Increase Physical Activity in Girls: The "Dads And Daughters Exercising and Empowered" (DADEE) Randomized Controlled Trial. *Annals of Behavioral Medicine*, 53(1):39–52, January 2019.

[25] World Health Organization. *Global recommendations on physical activity for health*. 2010.

[26] F. J. Penedo and J. R. Dahn. Exercise and well-being: a review of mental and physical health benefits associated with physical activity. *Current Opinion in Psychiatry*, 18(2):189–193, 2005.

[27] L. M. Wankel and B. G. Berger. The psychological and social benefits of sport and physical activity. *J. of Leisure Research*, 22(2):167–182, 1990.